



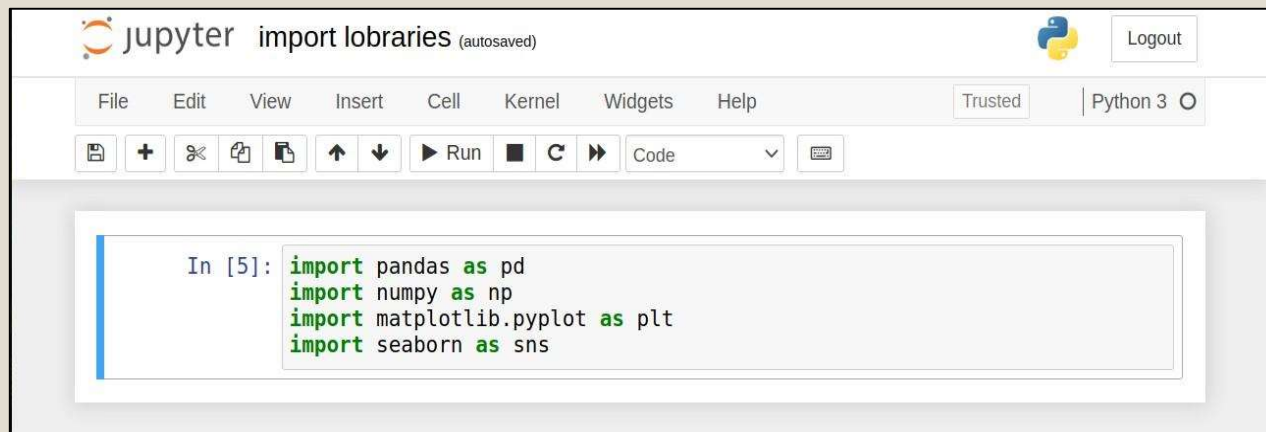
NUMPY

Installing Libraries

If you installed Anaconda, you do not need to download any libraries as it automatically installs all the popular data science libraries such as Pandas, Numpy, Matplotlib, Seaborn, etc.

Importing Libraries

- Open your Jupyter Notebook.
- To import a library we use the keyword **import** followed by library name.
- We can use the **as** keyword to use abbreviations for our library names.
- The common abbreviations used are
 - pd for pandas
 - np for numpy
 - plt for matplotlib.pyplot
 - sns for seaborn

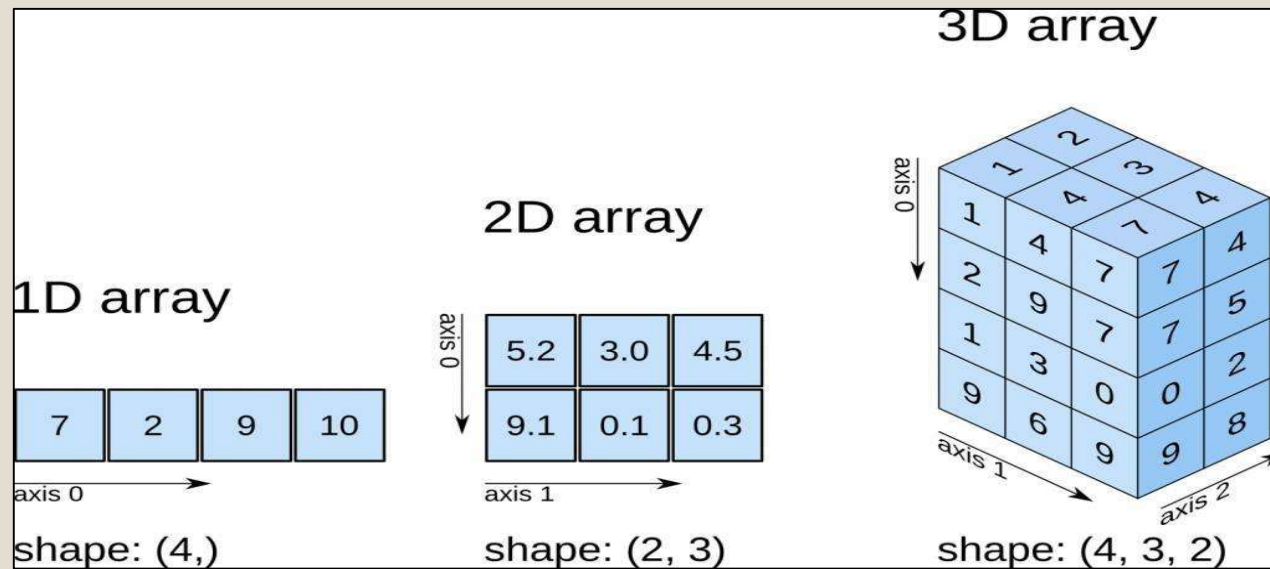


The screenshot shows a Jupyter Notebook window titled "jupyter import lobararies (autosaved)". The interface includes a top bar with the Jupyter logo, a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help), and a toolbar with icons for saving, adding, deleting, and running code. The main area contains a code cell with the following Python code:

```
In [5]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

NumPy Library

- NumPy stands for Numerical Python.
- It provides a data structure called NumPy array, which is a grid of values.
- It also provides a collection of high-level mathematical functions which can be performed on multi-dimensional NumPy arrays.



NumPy Arrays

What are NumPy Arrays

- NumPy array is a multidimensional data structure designed to handle large data sets easily.
- A NumPy array is called **ndarray**.
- We can find the number of dimensions of a NumPy array using **.ndim**.

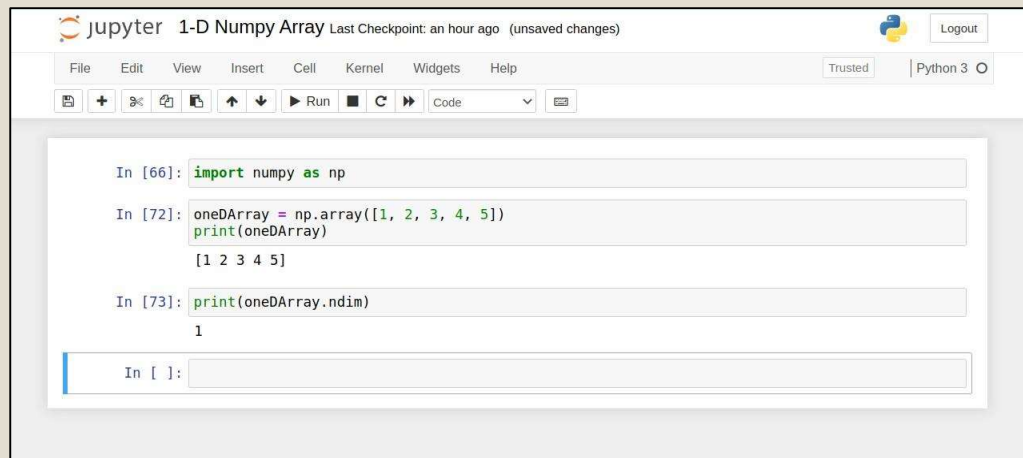
NumPy Arrays vs Python Lists

- NumPy arrays provide more built-in functionality as compared to Python lists.
- NumPy arrays make working with huge multi-dimensional data sets much easier with fewer syntax.
- NumPy arrays are also more efficient than Python lists in terms of memory consumption and speed.

Creating NumPy Arrays

1-D NumPy Arrays

- A 1-D NumPy array is where each element of the outermost array is a 0-D array (scalar).
- We can create a NumPy array using the `array()` function in the NumPy library.
- We can create a NumPy array using either Python lists or tuples.
- To create a 1-D NumPy Array, we provide a 1-D Python list or tuple to the `array()` function.



The image shows a Jupyter Notebook window titled "1-D Numpy Array". The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help), a toolbar with icons for file operations and execution, and a code editor. The code editor contains three input cells:


```
In [66]: import numpy as np
```

```
In [72]: oneDArray = np.array([1, 2, 3, 4, 5])
         print(oneDArray)
         [1 2 3 4 5]
```

```
In [73]: print(oneDArray.ndim)
         1
```

Below the code cells is an empty input cell labeled "In []:".

```
In [66]: import numpy as np
```

argument 

```
In [72]: oneDArray = np.array([1, 2, 3, 4, 5])
print(oneDArray)

[1 2 3 4 5]
```

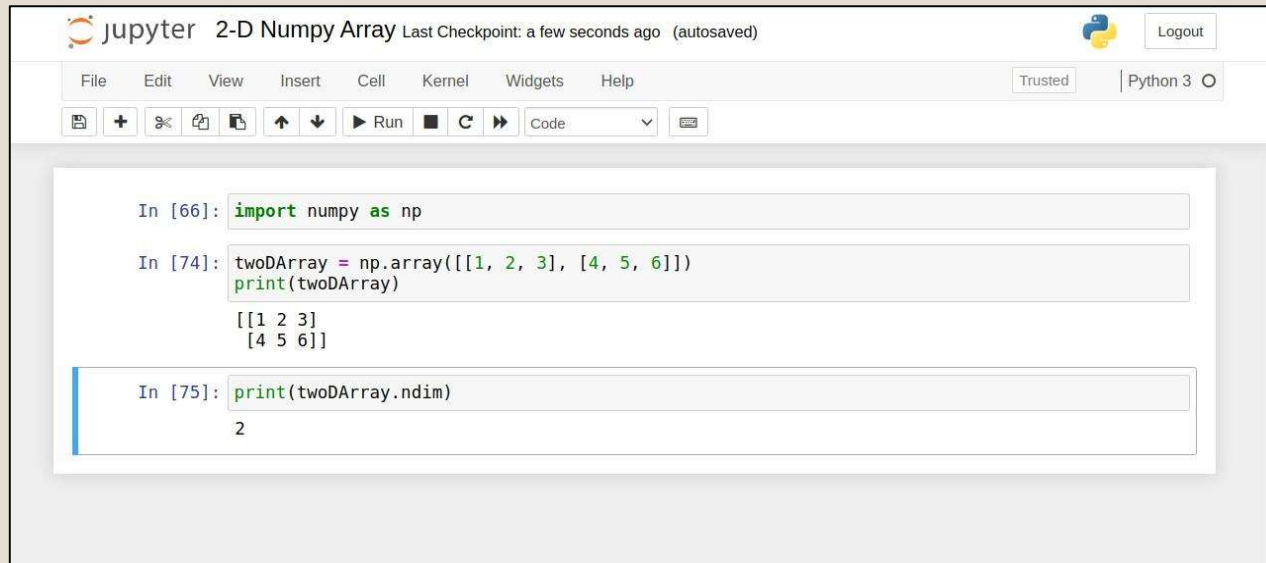
```
In [73]: print(oneDArray.ndim)

1
```

Creating NumPy Arrays

2-D NumPy Arrays

- A 2-D NumPy array is where each element in the outermost array is a 1-D array.
- To create a 2-D NumPy Array, we provide a 2-D Python list or tuple to the `array()` function.

A screenshot of a Jupyter Notebook interface. The title bar shows 'jupyter 2-D Numpy Array' and 'Last Checkpoint: a few seconds ago (autosaved)'. The menu bar includes 'File', 'Edit', 'View', 'Insert', 'Cell', 'Kernel', 'Widgets', and 'Help'. The toolbar contains icons for saving, adding cells, undo, redo, and running code. The code area shows three input cells. The first cell imports numpy as np. The second cell creates a 2-D array 'twoDArray' from a list of lists and prints it, showing the output as a 2x3 array. The third cell prints the number of dimensions of 'twoDArray', which is 2.

```
In [66]: import numpy as np

In [74]: twoDArray = np.array([[1, 2, 3], [4, 5, 6]])
         print(twoDArray)
[[1 2 3]
 [4 5 6]]

In [75]: print(twoDArray.ndim)
2
```



```
In [66]: import numpy as np
```

```
In [74]: twoDArray = np.array([[1, 2, 3], [4, 5, 6]])  
print(twoDArray)
```

```
[[1 2 3]  
 [4 5 6]]
```

1D

1D

```
In [75]: print(twoDArray.ndim)
```

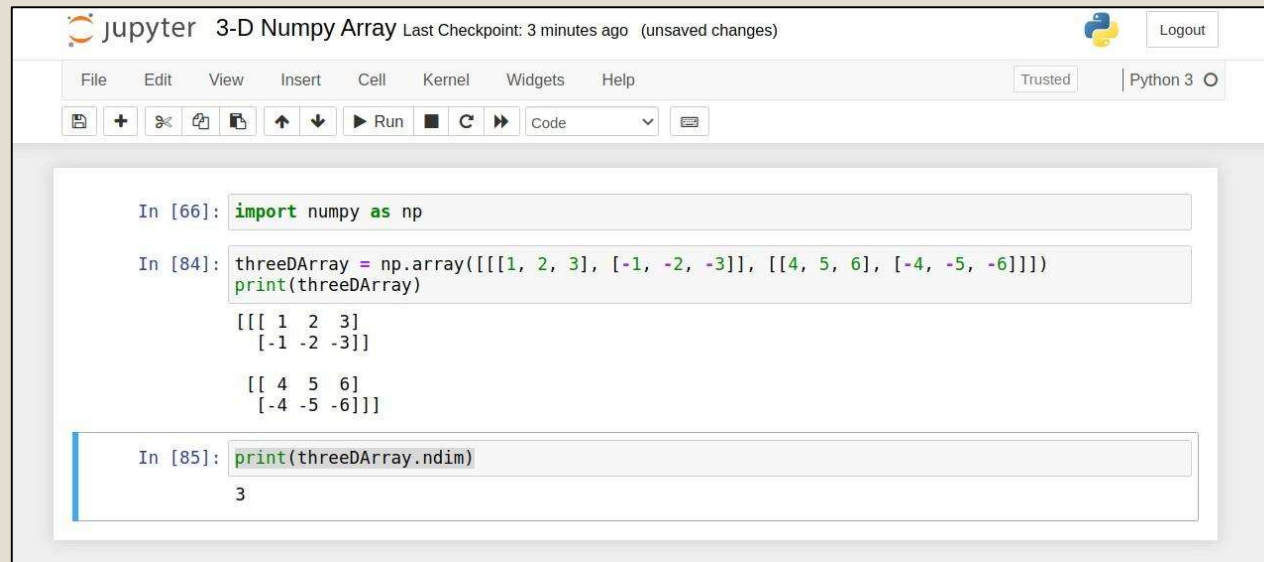
2

argument

Creating NumPy Arrays

3-D NumPy Arrays

- A 3-D NumPy array is where each element of the outermost array is a 2-D array.
- To create a 3-D NumPy Array, we provide a 3-D Python list or tuple to the array() function.



The image shows a Jupyter Notebook interface with the title "3-D Numpy Array". The notebook contains three code cells. The first cell imports numpy as np. The second cell creates a 3-D array named 'threeDArray' from a nested list and prints it. The output shows a 3x2x3 array. The third cell prints the number of dimensions of 'threeDArray', which is 3.

```
In [66]: import numpy as np

In [84]: threeDArray = np.array([[[1, 2, 3], [-1, -2, -3]], [[4, 5, 6], [-4, -5, -6]]])
         print(threeDArray)

[[[ 1  2  3]
  [-1 -2 -3]]

  [[ 4  5  6]
  [-4 -5 -6]]]

In [85]: print(threeDArray.ndim)

3
```

3D

```
[66]: import numpy as np
```

```
[84]: threeDArray = np.array([[[1, 2, 3], [-1, -2, -3]], [[4, 5, 6], [-4, -5, -6]]])  
print(threeDArray)
```

```
[[[ 1  2  3]  
  [-1 -2 -3]]
```

2D

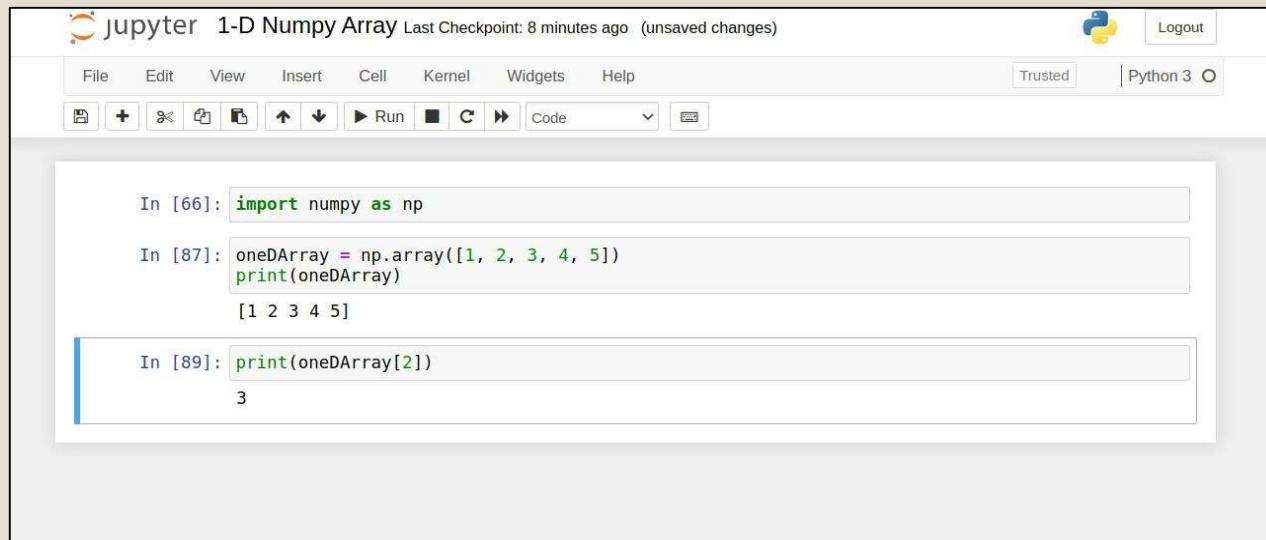
```
[[ 4  5  6]  
 [-4 -5 -6]]]
```

2D

Indexing NumPy Arrays

1-D NumPy Arrays

- Indexing a 1-D NumPy array is the same as indexing a 1-D Python list.
- Provide the index of the element inside the square brackets to get that element.

A screenshot of a Jupyter Notebook interface. The title bar shows 'jupyter 1-D Numpy Array' and 'Last Checkpoint: 8 minutes ago (unsaved changes)'. The menu bar includes 'File', 'Edit', 'View', 'Insert', 'Cell', 'Kernel', 'Widgets', and 'Help'. The toolbar contains icons for saving, adding, deleting, copying, pasting, undo, redo, and running code. The code area shows three input cells. The first cell contains 'import numpy as np'. The second cell contains 'oneDArray = np.array([1, 2, 3, 4, 5])' and 'print(oneDArray)', with the output '[1 2 3 4 5]' displayed below. The third cell contains 'print(oneDArray[2])' and the output '3' is displayed below it.

```
In [66]: import numpy as np

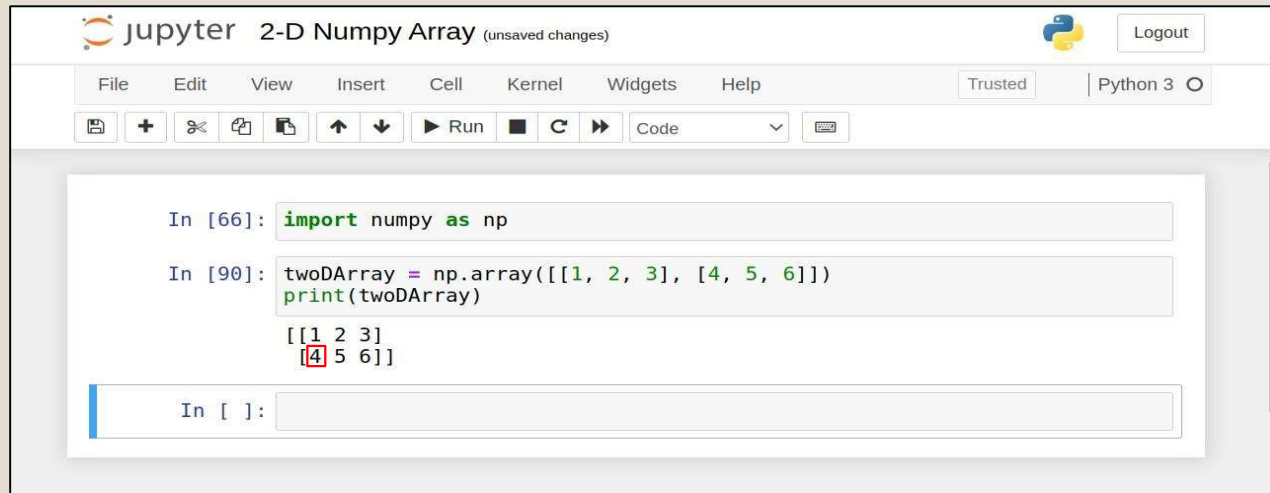
In [87]: oneDArray = np.array([1, 2, 3, 4, 5])
         print(oneDArray)
         [1 2 3 4 5]

In [89]: print(oneDArray[2])
         3
```

Indexing NumPy Arrays

2-D NumPy Arrays

- To index a 2-D NumPy array, we provide 2 values inside the square brackets ([]).
 - First value is the index of the inner array
 - Second value is the index of the element inside the inner array
- In the following example, we get the first element of the second array.

A screenshot of a Jupyter Notebook interface. The title bar says "jupyter 2-D Numpy Array (unsaved changes)". The menu bar includes File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. Below the menu bar is a toolbar with icons for saving, adding cells, undo, redo, and running code. The notebook area shows two code cells. The first cell, labeled "In [66]:", contains the code "import numpy as np". The second cell, labeled "In [90]:", contains the code "twoDArray = np.array([[1, 2, 3], [4, 5, 6]])" followed by "print(twoDArray)". The output of the second cell is a 2x3 NumPy array: [[1 2 3], [4 5 6]]. The element '4' in the second row, first column is highlighted with a red square. Below the output is an empty code cell labeled "In []:".

```
jupyter 2-D Numpy Array (unsaved changes) Logout
```

```
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3
```

```
In [66]: import numpy as np
```

```
In [90]: twoDArray = np.array([[1, 2, 3], [4, 5, 6]])
          print(twoDArray)
```

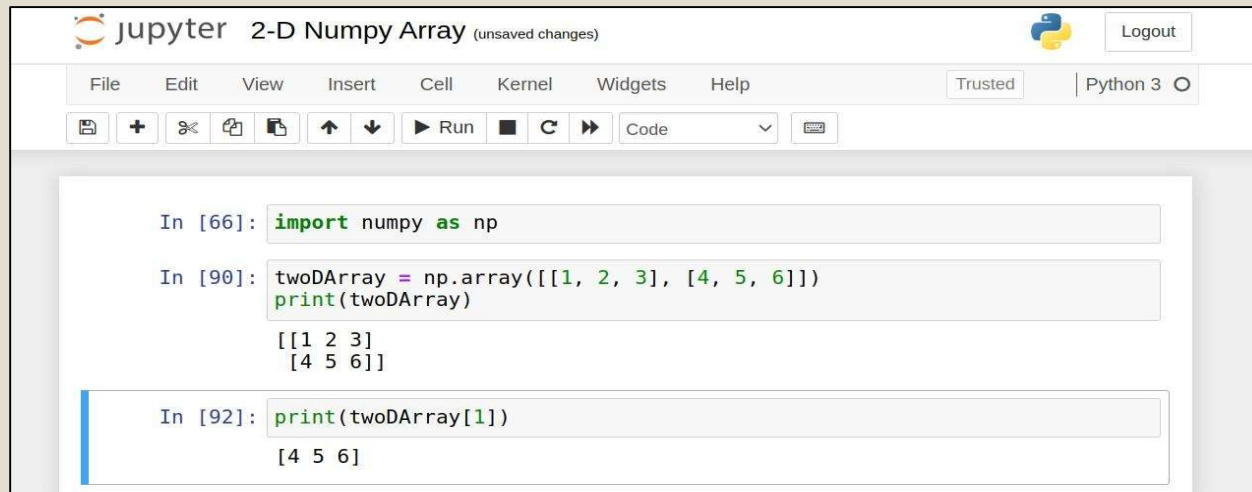
```
[[1 2 3]
 [4 5 6]]
```

```
In [ ]:
```

Indexing NumPy Arrays

2-D NumPy Arrays

- The first dimension contains 2 arrays.
- If we say `twoDArray[1]`, we get the second array.

A screenshot of a Jupyter Notebook interface. The title bar shows 'jupyter 2-D Numpy Array (unsaved changes)' and a 'Logout' button. The menu bar includes 'File', 'Edit', 'View', 'Insert', 'Cell', 'Kernel', 'Widgets', and 'Help'. Below the menu is a toolbar with icons for saving, adding cells, deleting, copying, pasting, undo, redo, and running code. The notebook contains three code cells. The first cell imports numpy as np. The second cell creates a 2D array 'twoDArray' with values [[1, 2, 3], [4, 5, 6]] and prints it, showing the output as a 2x3 array. The third cell prints 'twoDArray[1]', showing the output as the second row of the array, [4 5 6].

```
In [66]: import numpy as np

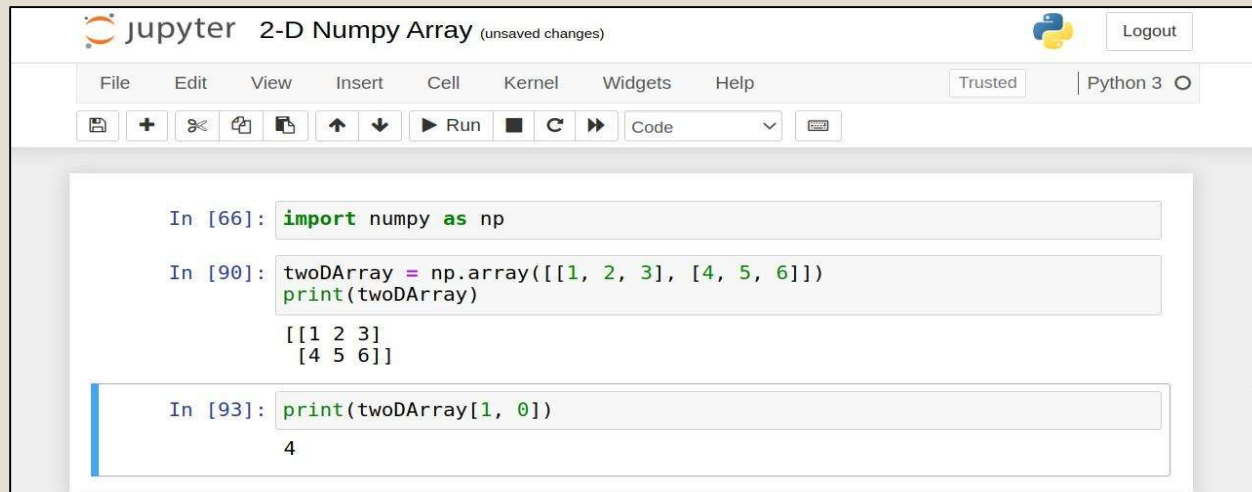
In [90]: twoDArray = np.array([[1, 2, 3], [4, 5, 6]])
         print(twoDArray)
[[1 2 3]
 [4 5 6]]

In [92]: print(twoDArray[1])
[4 5 6]
```

Indexing NumPy Arrays

2-D NumPy Arrays

- The second dimension contains 3 elements.
- If we say `twoDArray[1, 0]`, we get the first element of the second array.

A screenshot of a Jupyter Notebook interface. The title bar says "jupyter 2-D Numpy Array (unsaved changes)". The menu bar includes File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. The toolbar has icons for saving, adding cells, undo, redo, and running code. The code area shows three input cells. The first cell imports numpy as np. The second cell creates a 2D array and prints it, showing the output as a 2x3 array. The third cell prints the element at index [1, 0], showing the output as 4.

```
In [66]: import numpy as np

In [90]: twoDArray = np.array([[1, 2, 3], [4, 5, 6]])
         print(twoDArray)

         [[1 2 3]
          [4 5 6]]

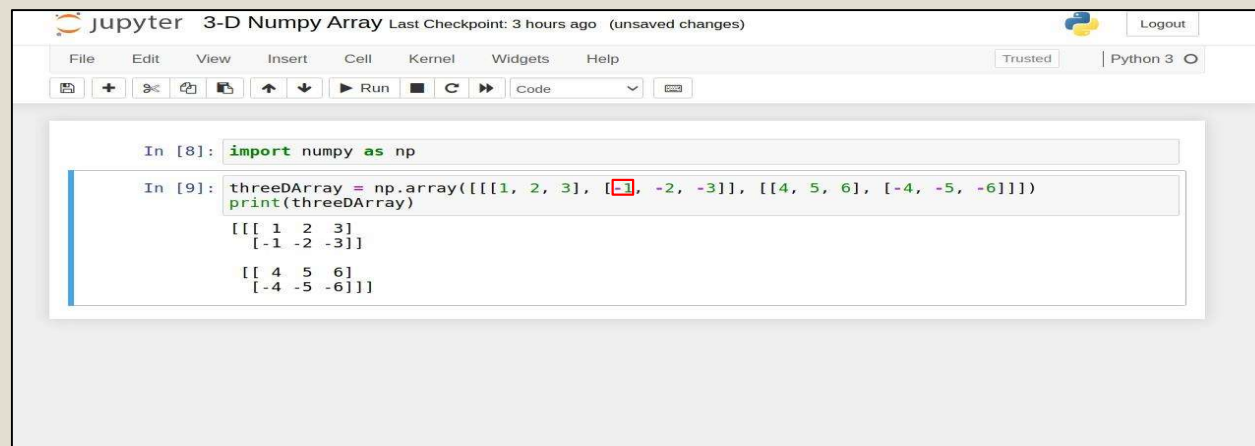
In [93]: print(twoDArray[1, 0])

         4
```

Indexing NumPy Arrays

3-D NumPy Arrays

- To index a 3-D NumPy array, we provide 3 values inside the square brackets ([]).
 - First value is the index of the inner 2-D array in the first dimension.
 - Second value is the index of the inner 1-D array in the second dimension.
 - Third value is the index of the element in the third dimension.
- In the following example, we get the first element of the second array of the first array.

A screenshot of a Jupyter Notebook interface. The title bar says 'jupyter 3-D Numpy Array Last Checkpoint: 3 hours ago (unsaved changes)'. The menu bar includes File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. Below the menu is a toolbar with icons for file operations and execution. The code area shows two input cells. The first cell (In [8]) contains 'import numpy as np'. The second cell (In [9]) contains 'threeDArray = np.array([[[1, 2, 3], [-1, -2, -3]], [[4, 5, 6], [-4, -5, -6]]])' followed by 'print(threeDArray)'. The output of the second cell is displayed below the code: '[[[1 2 3] [-1 -2 -3]] [[4 5 6] [-4 -5 -6]]]'. A red box highlights the value '-1' in the code, which corresponds to the first element of the second array in the first dimension of the output.

```
In [8]: import numpy as np

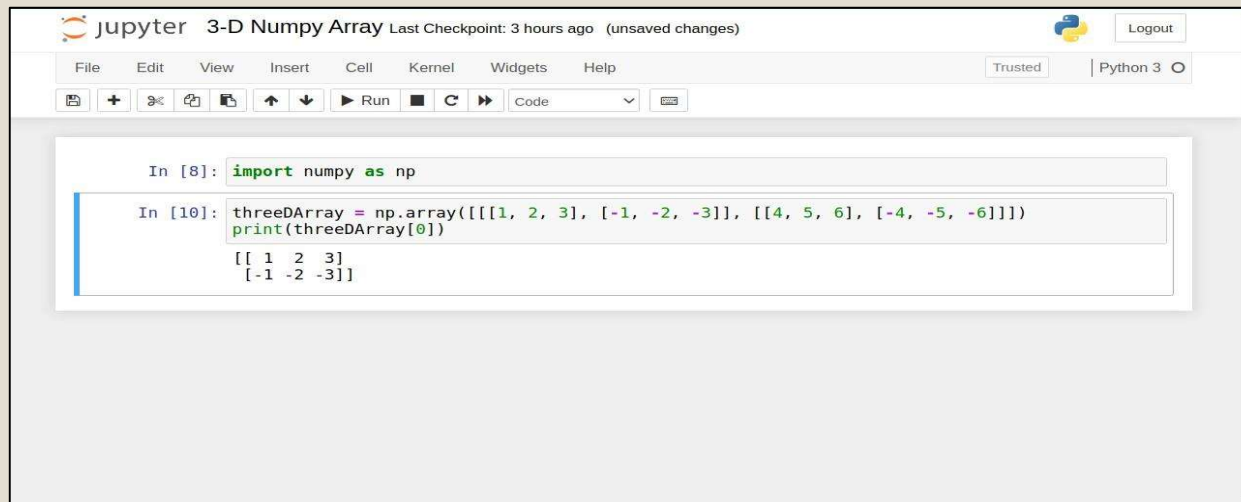
In [9]: threeDArray = np.array([[[1, 2, 3], [-1, -2, -3]], [[4, 5, 6], [-4, -5, -6]]])
print(threeDArray)

[[[ 1 2 3]
  [-1 -2 -3]]
 [[ 4 5 6]
  [-4 -5 -6]]]
```


Indexing NumPy Arrays

3-D NumPy Arrays

- The first dimension contains 2 arrays.
- If we say `threeDArray[0]`, we get the first array.

A screenshot of a Jupyter Notebook interface. The title bar shows 'jupyter 3-D Numpy Array' and 'Last Checkpoint: 3 hours ago (unsaved changes)'. The menu bar includes 'File', 'Edit', 'View', 'Insert', 'Cell', 'Kernel', 'Widgets', and 'Help'. The toolbar contains icons for file operations, running, and other notebook functions. The code area shows two input cells. The first cell, labeled 'In [8]:', contains the code 'import numpy as np'. The second cell, labeled 'In [10]:', contains the code 'threeDArray = np.array([[[1, 2, 3], [-1, -2, -3]], [[4, 5, 6], [-4, -5, -6]]])' followed by 'print(threeDArray[0])'. The output of the second cell is displayed below the code: a 2x3 array with values [[1 2 3], [-1 -2 -3]].

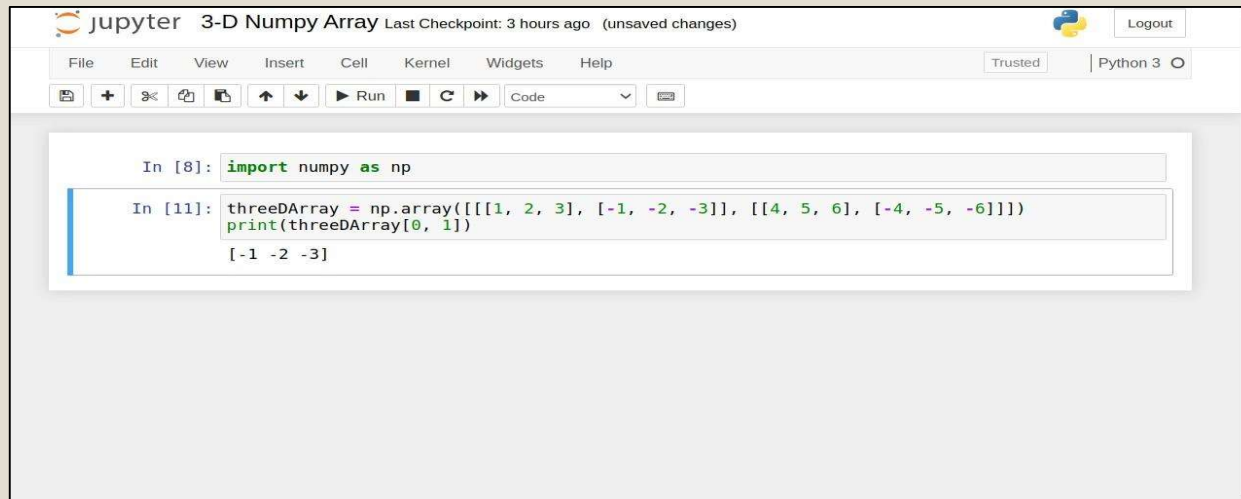
```
In [8]: import numpy as np

In [10]: threeDArray = np.array([[[1, 2, 3], [-1, -2, -3]], [[4, 5, 6], [-4, -5, -6]]])
         print(threeDArray[0])
         [[ 1  2  3]
          [-1 -2 -3]]
```

Indexing NumPy Arrays

3-D NumPy Arrays

- The second dimension again contains 2 arrays.
- If we say `threeDArray[0, 1]`, we get the second array of the first array.

A screenshot of a Jupyter Notebook interface. The title bar shows 'jupyter 3-D Numpy Array' and 'Last Checkpoint: 3 hours ago (unsaved changes)'. The menu bar includes 'File', 'Edit', 'View', 'Insert', 'Cell', 'Kernel', 'Widgets', and 'Help'. The toolbar contains icons for file operations, running, and code execution. The code area shows two input cells. The first cell, labeled 'In [8]:', contains the code 'import numpy as np'. The second cell, labeled 'In [11]:', contains the code 'threeDArray = np.array([[[1, 2, 3], [-1, -2, -3]], [[4, 5, 6], [-4, -5, -6]]])' followed by 'print(threeDArray[0, 1])'. The output of the second cell is displayed as '[-1 -2 -3]'.

```
jupyter 3-D Numpy Array Last Checkpoint: 3 hours ago (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3
+ %< > >> Run [ ] Code

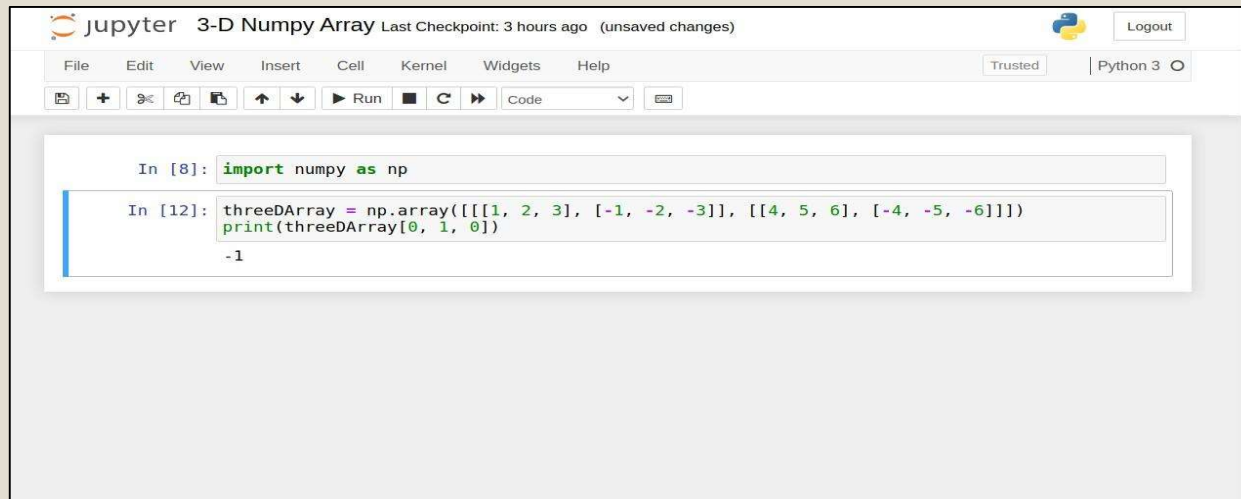
In [8]: import numpy as np

In [11]: threeDArray = np.array([[[1, 2, 3], [-1, -2, -3]], [[4, 5, 6], [-4, -5, -6]]])
         print(threeDArray[0, 1])
         [-1 -2 -3]
```

Indexing NumPy Arrays

3-D NumPy Arrays

- The third dimension contains 3 values.
- If we say `threeDArray[0, 1, 0]`, we get the first element of the second array of the first array.

A screenshot of a Jupyter Notebook interface. The title bar shows 'jupyter 3-D Numpy Array' and 'Last Checkpoint: 3 hours ago (unsaved changes)'. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running, and code execution. The code area contains two input cells. The first cell, labeled 'In [8]:', contains the code 'import numpy as np'. The second cell, labeled 'In [12]:', contains the code 'threeDArray = np.array([[[1, 2, 3], [-1, -2, -3]], [[4, 5, 6], [-4, -5, -6]]])' followed by 'print(threeDArray[0, 1, 0])'. The output of the second cell is '-1'.

```
jupyter 3-D Numpy Array Last Checkpoint: 3 hours ago (unsaved changes)

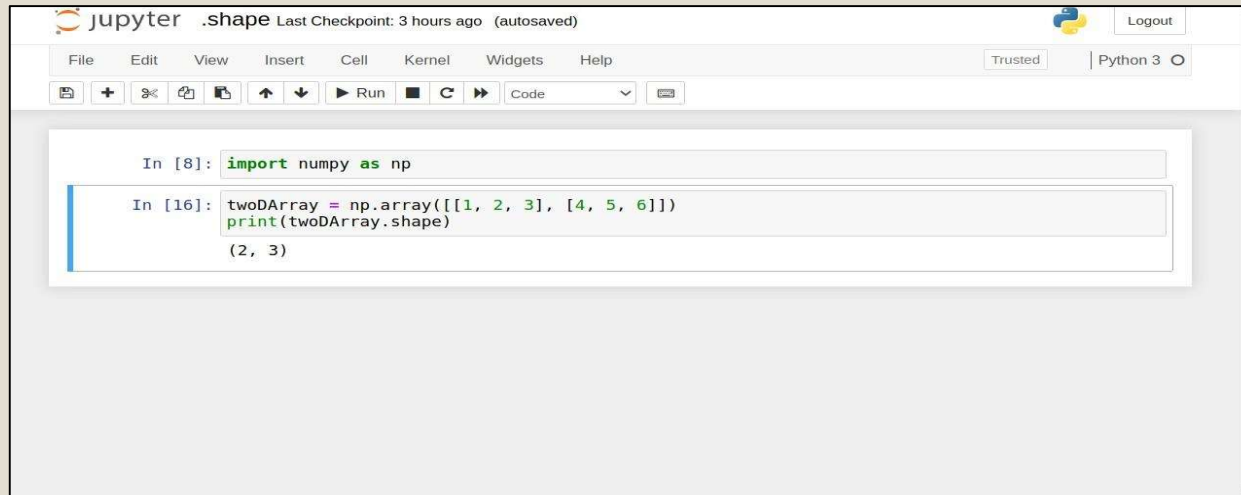
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [8]: import numpy as np

In [12]: threeDArray = np.array([[[1, 2, 3], [-1, -2, -3]], [[4, 5, 6], [-4, -5, -6]]])
         print(threeDArray[0, 1, 0])
         -1
```

Array Shape

- NumPy arrays have a shape attribute which returns a tuple.
 - First value of the tuple gives the number of dimensions in the array
 - Second value of the tuple gives the number of elements in each dimension

A screenshot of a Jupyter Notebook interface. The top bar shows the Jupyter logo, the filename ".shape", and a status message "Last Checkpoint: 3 hours ago (autosaved)". On the right, there is a "Logout" button and a Python 3 logo. Below the top bar is a menu bar with "File", "Edit", "View", "Insert", "Cell", "Kernel", "Widgets", and "Help". A toolbar contains icons for file operations, a "Run" button, and a "Code" dropdown menu. The main area displays two code cells. The first cell, labeled "In [8]:", contains the code "import numpy as np". The second cell, labeled "In [16]:", contains the code "twoDArray = np.array([[1, 2, 3], [4, 5, 6]])" followed by "print(twoDArray.shape)". The output of the second cell is "(2, 3)".

```
jupyter .shape Last Checkpoint: 3 hours ago (autosaved) Logout
```

```
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3
```

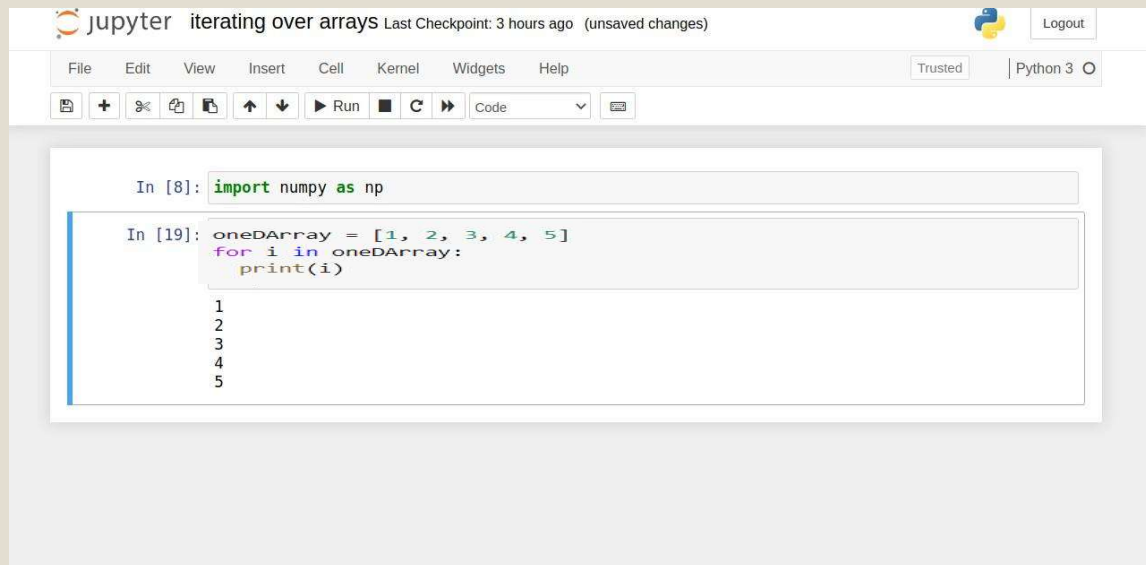
```
In [8]: import numpy as np
```

```
In [16]: twoDArray = np.array([[1, 2, 3], [4, 5, 6]])
          print(twoDArray.shape)
          (2, 3)
```

Iterating Over NumPy Arrays

1-D NumPy Arrays

- We can use a for loop to iterate over a 1-D array just as we do in a 1-D Python list.



The image shows a Jupyter Notebook interface with the title "iterating over arrays". The notebook has two code cells. The first cell, labeled "In [8]:", contains the code `import numpy as np`. The second cell, labeled "In [19]:", contains the code `oneDArray = [1, 2, 3, 4, 5]`, `for i in oneDArray:`, and `print(i)`. The output of the second cell is displayed below the code, showing the numbers 1, 2, 3, 4, and 5 on separate lines. The Jupyter interface includes a menu bar with options like File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. There is also a toolbar with icons for running, saving, and other notebook functions. The status bar at the bottom indicates "Trusted" and "Python 3".

```
In [8]: import numpy as np

In [19]: oneDArray = [1, 2, 3, 4, 5]
         for i in oneDArray:
             print(i)

1
2
3
4
5
```

Iterating Over NumPy Arrays

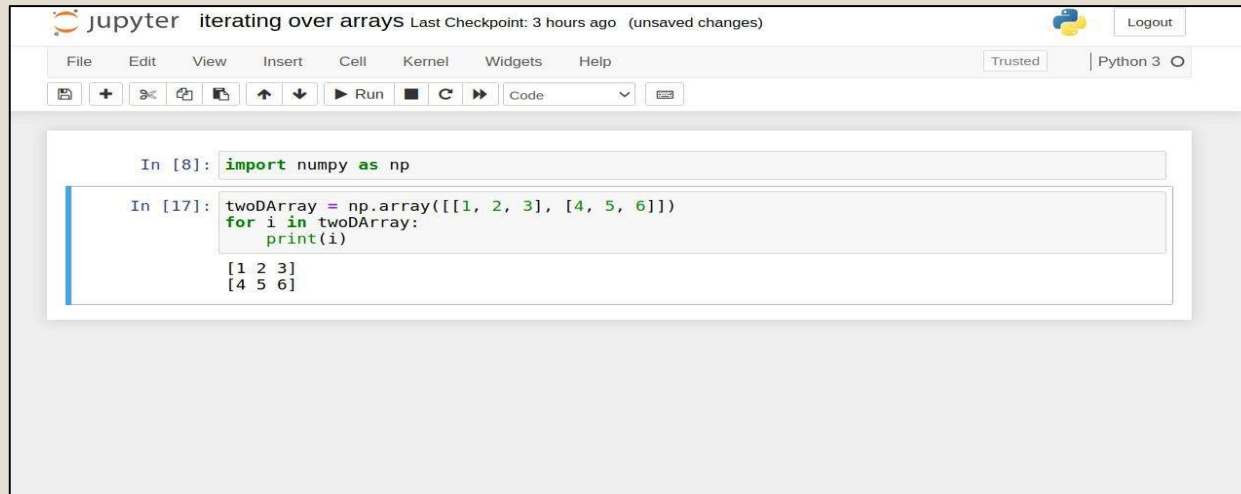
2-D NumPy Arrays

- We can use a nested for loop to iterate over a 2-D array.
 - The outer for loop iterates over the outer array.
 - The inner for loop iterates over the inner array.

Iterating Over NumPy Arrays

2-D NumPy Arrays

- We use a for loop to iterate over the outer array.
- We print all the inner arrays.

A screenshot of a Jupyter Notebook interface. The title bar shows 'jupyter iterating over arrays' with a 'Last Checkpoint: 3 hours ago' and '(unsaved changes)' status. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running, and code execution. The code area contains two input cells. The first cell, labeled 'In [8]:', contains the code 'import numpy as np'. The second cell, labeled 'In [17]:', contains the code 'twoDArray = np.array([[1, 2, 3], [4, 5, 6]])', followed by a for loop 'for i in twoDArray:' and 'print(i)'. The output of the second cell is displayed below the code, showing two lines: '[1 2 3]' and '[4 5 6]'. The interface also shows a 'Trusted' status and 'Python 3' as the selected kernel.

```
In [8]: import numpy as np

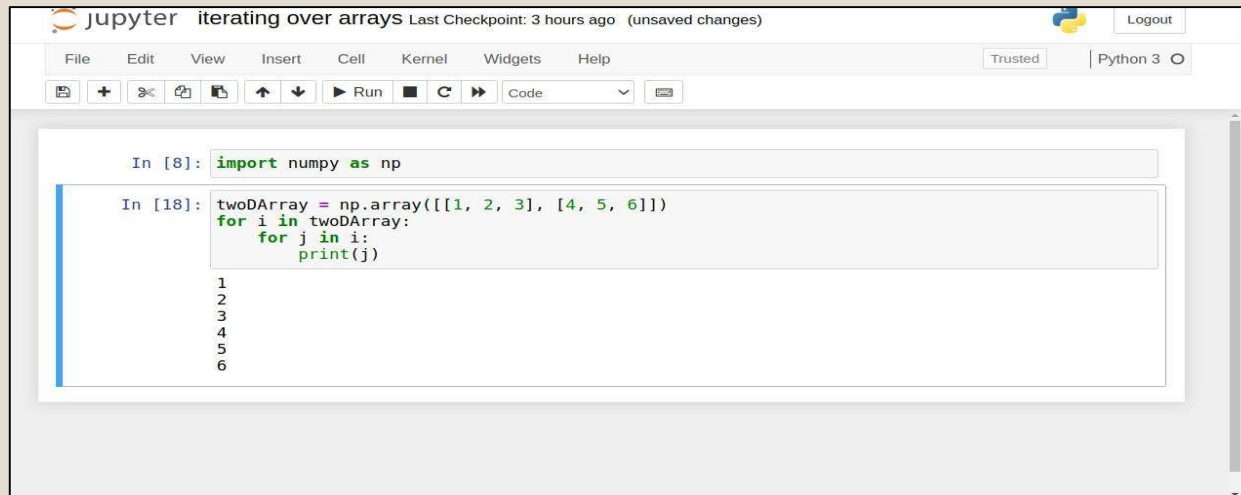
In [17]: twoDArray = np.array([[1, 2, 3], [4, 5, 6]])
         for i in twoDArray:
             print(i)

[1 2 3]
[4 5 6]
```

Iterating Over NumPy Arrays

2-D NumPy Arrays

- We use another for loop nested inside the outer for loop to iterate over the inner array.
- We print all the elements in each of the inner arrays.

A screenshot of a Jupyter Notebook interface. The title bar shows 'jupyter iterating over arrays' and 'Last Checkpoint: 3 hours ago (unsaved changes)'. The menu bar includes 'File', 'Edit', 'View', 'Insert', 'Cell', 'Kernel', 'Widgets', and 'Help'. The toolbar contains icons for file operations, running, and code execution. The code area shows two input cells. The first cell contains 'In [8]: import numpy as np'. The second cell contains 'In [18]: twoDArray = np.array([[1, 2, 3], [4, 5, 6]])' followed by a nested for loop: 'for i in twoDArray:', 'for j in i:', and 'print(j)'. The output of the second cell shows the numbers 1, 2, 3, 4, 5, and 6 printed on separate lines.

```
In [8]: import numpy as np

In [18]: twoDArray = np.array([[1, 2, 3], [4, 5, 6]])
         for i in twoDArray:
             for j in i:
                 print(j)

1
2
3
4
5
6
```


Iterating Over NumPy Arrays

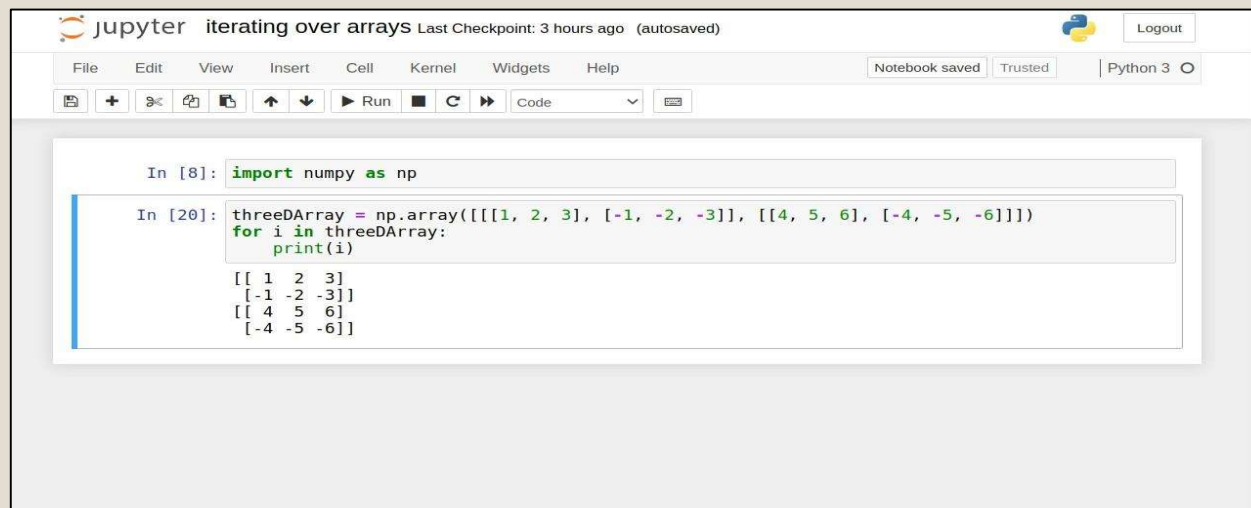
3-D NumPy Arrays

- We can use 3 nested for loops to iterate over a 3-D array.
 - The outermost for loop iterates over the arrays in the first dimension.
 - The middle for loop iterates over the arrays in the second dimension.
 - The innermost for loop iterates over all the elements in the third dimension.

Iterating Over NumPy Arrays

3-D NumPy Arrays

- Outermost array contains 2 arrays both of which are 2-D.
- We use a for loop to print these 2-D arrays.



The image shows a Jupyter Notebook interface with the title "iterating over arrays" and a status bar indicating "Last Checkpoint: 3 hours ago (autosaved)". The notebook is running on Python 3. The code in the notebook is as follows:

```
In [8]: import numpy as np

In [20]: threeDArray = np.array([[[1, 2, 3], [-1, -2, -3]], [[4, 5, 6], [-4, -5, -6]]])
         for i in threeDArray:
             print(i)
```

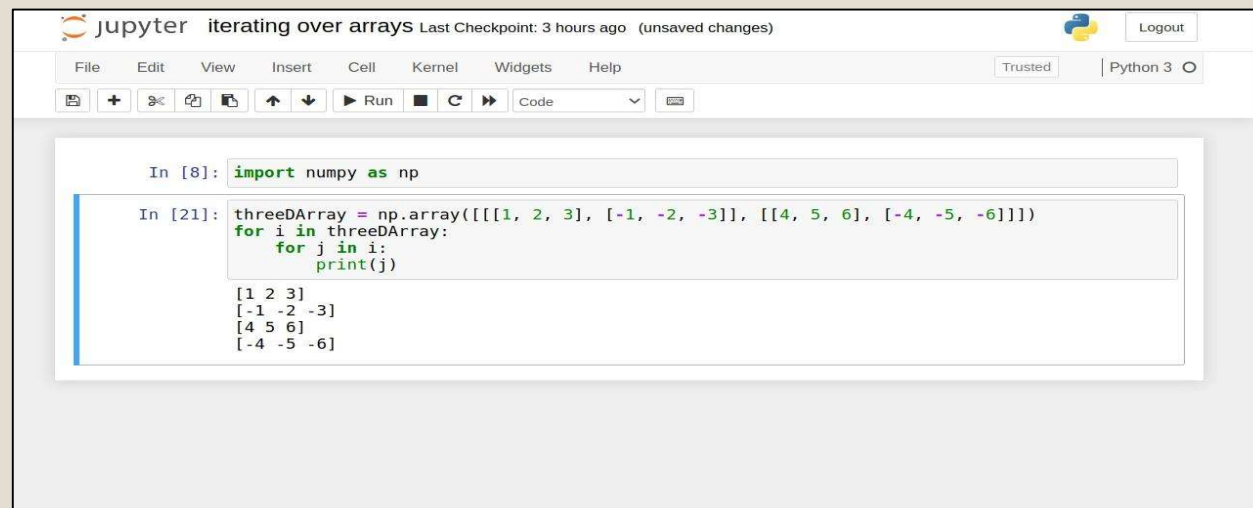
The output of the code is:

```
[[ 1  2  3]
 [-1 -2 -3]]
[[ 4  5  6]
 [-4 -5 -6]]
```

Iterating Over NumPy Arrays

3-D NumPy Arrays

- Each of the 2-D array contains 2 arrays in the second dimension, each of which is 1-D.
- We use another for loop nested within the first for loop to print these 1-D arrays.



The image shows a Jupyter Notebook window titled "iterating over arrays" with a "Last Checkpoint: 3 hours ago (unsaved changes)" status. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running, and code execution. The code area contains two input cells. The first cell, labeled "In [8]:", contains the code `import numpy as np`. The second cell, labeled "In [21]:", contains the code to create a 3D array and iterate over it. The output of the second cell is displayed below the code, showing four 1D arrays: `[1 2 3]`, `[-1 -2 -3]`, `[4 5 6]`, and `[-4 -5 -6]`.

```
In [8]: import numpy as np

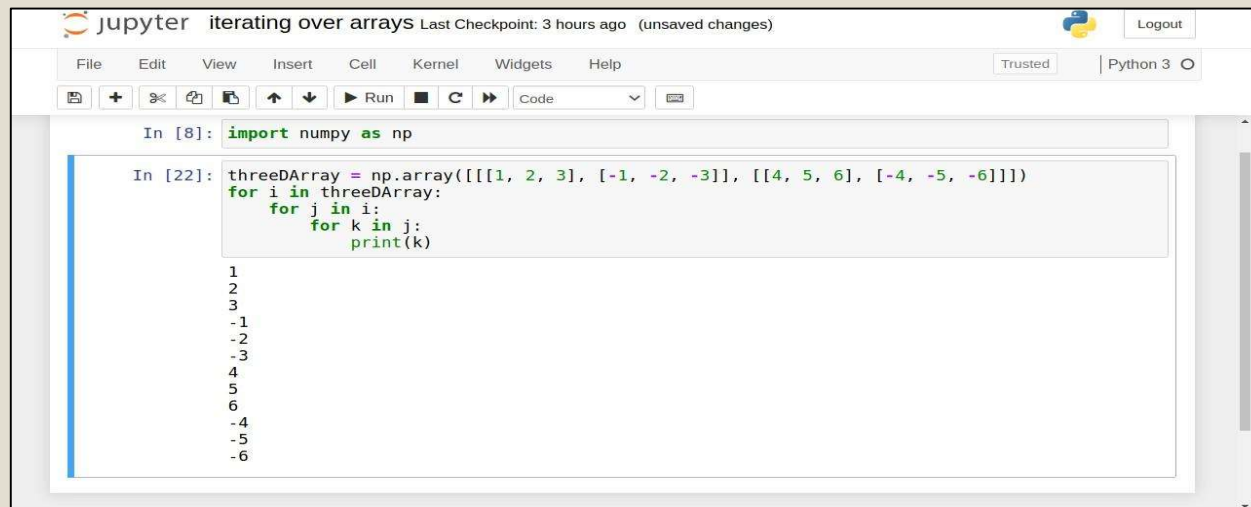
In [21]: threeDArray = np.array([[[1, 2, 3], [-1, -2, -3]], [[4, 5, 6], [-4, -5, -6]]])
         for i in threeDArray:
             for j in i:
                 print(j)

         [1 2 3]
         [-1 -2 -3]
         [4 5 6]
         [-4 -5 -6]
```

Iterating Over NumPy Arrays

3-D NumPy Arrays

- Each of the 1-D array contains 3 elements in the third dimension, each of which is 1-D.
- We use another for loop nested within the first 2 for loops to print these elements.

A screenshot of a Jupyter Notebook interface. The title bar reads "jupyter iterating over arrays" with a "Last Checkpoint: 3 hours ago (unsaved changes)" status and a "Logout" button. The menu bar includes File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. Below the menu is a toolbar with icons for file operations, running, and code execution. The notebook contains two code cells. The first cell, labeled "In [8]:", contains the code "import numpy as np". The second cell, labeled "In [22]:", contains a 3D NumPy array definition and a nested loop for iteration. The array is defined as "threeDArray = np.array([[[1, 2, 3], [-1, -2, -3]], [[4, 5, 6], [-4, -5, -6]]])". The nested loops are "for i in threeDArray:", "for j in i:", and "for k in j:", with "print(k)" inside the innermost loop. The output of the second cell shows the values 1, 2, 3, -1, -2, -3, 4, 5, 6, -4, -5, -6 printed on separate lines.

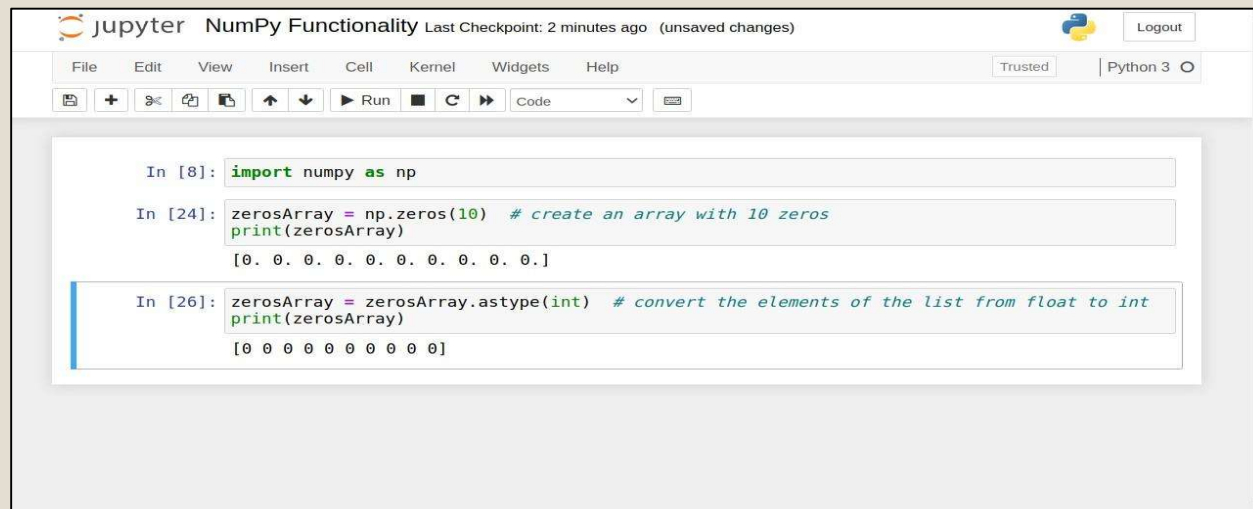
```
jupyter iterating over arrays Last Checkpoint: 3 hours ago (unsaved changes) Logout
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3
In [8]: import numpy as np
In [22]: threeDArray = np.array([[[1, 2, 3], [-1, -2, -3]], [[4, 5, 6], [-4, -5, -6]]])
         for i in threeDArray:
           for j in i:
             for k in j:
               print(k)
1
2
3
-1
-2
-3
4
5
6
-4
-5
-6
```

Mathematics for Data Science

- NumPy provides us with a huge collection of high-level functions for multi-dimensional arrays.
- Let's have a look at some of the functionality provided by NumPy.

.zeros()

- To create a NumPy array prefilled with zeros, we can use the `.zeros()` built-in NumPy function.
- `.zeros()` gives us a list prefilled with float zeros. To convert this list into integer list, we use the `.astype()` function.



The image shows a Jupyter Notebook interface with the title "NumPy Functionality". The interface includes a top bar with "File", "Edit", "View", "Insert", "Cell", "Kernel", "Widgets", and "Help" menus. Below the menus is a toolbar with icons for file operations, a "Run" button, and a "Code" dropdown. The notebook contains three code cells:

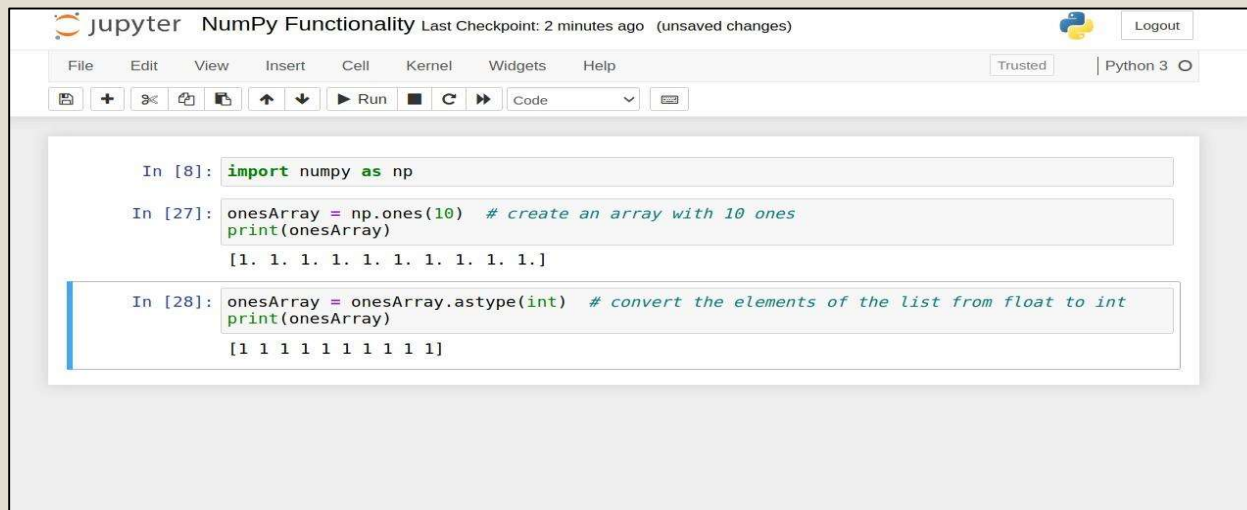
```
In [8]: import numpy as np
```

```
In [24]: zerosArray = np.zeros(10) # create an array with 10 zeros
print(zerosArray)
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

```
In [26]: zerosArray = zerosArray.astype(int) # convert the elements of the list from float to int
print(zerosArray)
[0 0 0 0 0 0 0 0 0 0]
```

.ones()

- To create a NumPy array prefilled with ones, we can use the `.ones()` built-in NumPy function.
- `.ones()` gives us a list prefilled with float ones. To convert this list into integer list, we use the `.astype()` function.

A screenshot of a Jupyter Notebook interface titled "NumPy Functionality". The interface includes a top menu bar with options like File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. Below the menu is a toolbar with icons for saving, adding cells, and running code. The notebook contains three code cells. The first cell imports NumPy as np. The second cell creates a NumPy array of 10 ones using np.ones(10) and prints it, showing the output as a list of float ones. The third cell converts the float array to an integer array using .astype(int) and prints it, showing the output as a list of integer ones.

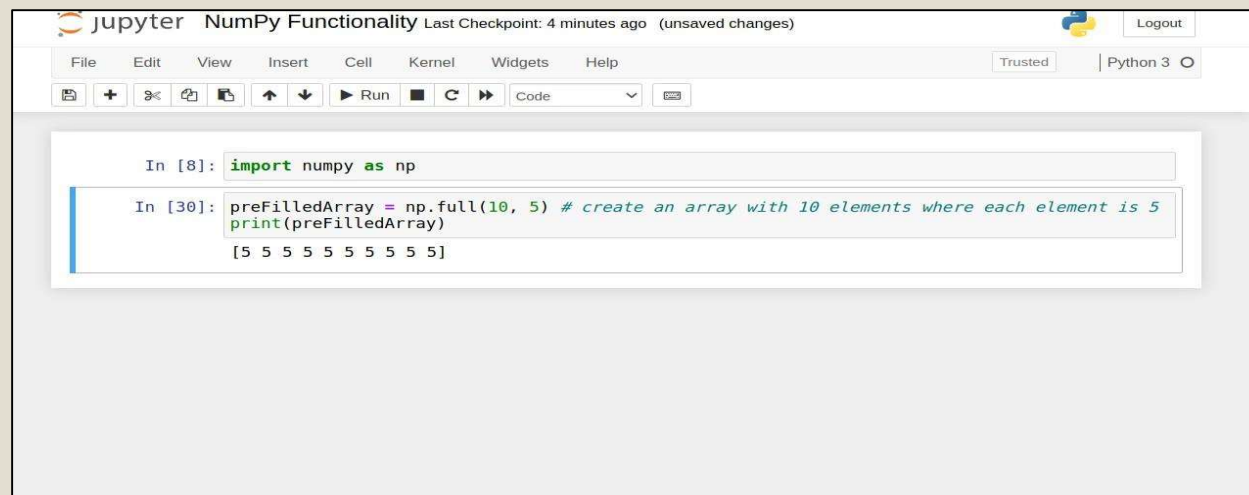
```
In [8]: import numpy as np

In [27]: onesArray = np.ones(10) # create an array with 10 ones
print(onesArray)
[1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]

In [28]: onesArray = onesArray.astype(int) # convert the elements of the list from float to int
print(onesArray)
[1 1 1 1 1 1 1 1 1 1]
```

.full()

- To create a NumPy array prefilled with some specific number, we can use the `.full()` built-in NumPy function.
 - First argument in the `.full()` function is the size of the array
 - Second argument in the `.full()` function is the value that we want our list to be pre-filled with

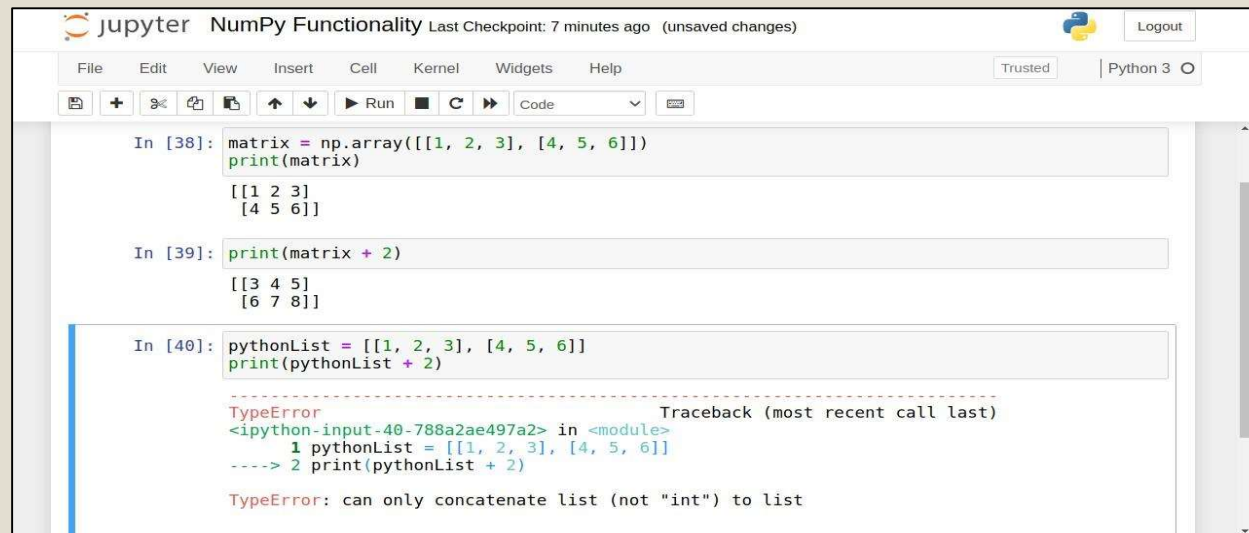
A screenshot of a Jupyter Notebook interface. The title bar reads "jupyter NumPy Functionality Last Checkpoint: 4 minutes ago (unsaved changes)". The menu bar includes File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. The toolbar shows icons for file operations, cell navigation, and execution. The code area contains two input cells. The first cell, labeled "In [8]:", contains the code `import numpy as np`. The second cell, labeled "In [30]:", contains the code `preFilledArray = np.full(10, 5) # create an array with 10 elements where each element is 5` followed by `print(preFilledArray)`. The output of the second cell is displayed as `[5 5 5 5 5 5 5 5 5 5]`.

```
jupyter NumPy Functionality Last Checkpoint: 4 minutes ago (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3
In [8]: import numpy as np
In [30]: preFilledArray = np.full(10, 5) # create an array with 10 elements where each element is 5
         print(preFilledArray)
         [5 5 5 5 5 5 5 5 5 5]
```


Scalar Operations

Addition

- We can add a scalar to a NumPy array simply by using the (+) operator.
- The scalar quantity is added to each of the elements of the array.
- Note that adding a scalar to a Python list will result in an error.



The screenshot shows a Jupyter Notebook interface with the title 'NumPy Functionality'. It contains three code cells. The first cell creates a NumPy array and prints it. The second cell adds 2 to the array and prints the result. The third cell attempts to add 2 to a Python list, which results in a `TypeError`.

```
jupyter NumPy Functionality Last Checkpoint: 7 minutes ago (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3
In [38]: matrix = np.array([[1, 2, 3], [4, 5, 6]])
         print(matrix)
         [[1 2 3]
          [4 5 6]]

In [39]: print(matrix + 2)
         [[3 4 5]
          [6 7 8]]

In [40]: pythonList = [[1, 2, 3], [4, 5, 6]]
         print(pythonList + 2)

         -----
         Traceback (most recent call last)
         <ipython-input-40-788a2ae497a2> in <module>
             1 pythonList = [[1, 2, 3], [4, 5, 6]]
         ----> 2 print(pythonList + 2)

         TypeError: can only concatenate list (not "int") to list
```

Scalar Operations

Subtraction

- We can subtract a scalar from a NumPy array simply by using the (-) operator.
- The scalar quantity is subtracted from each of the elements of the array.
- Note that subtracting a scalar from a Python list will result in an error.



The screenshot shows a Jupyter Notebook interface with the title 'jupyter NumPy Functionality'. The top bar includes a 'Logout' button and a 'Python 3' selector. The notebook contains three code cells. The first cell, labeled 'In [41]:', defines a NumPy array 'matrix' with values [[1, 2, 3], [4, 5, 6]] and prints it, resulting in the output:

```
[[1 2 3]
 [4 5 6]]
```

. The second cell, labeled 'In [42]:', prints 'matrix - 2', resulting in the output:

```
[[ -1  0  1]
 [ 2  3  4]]
```

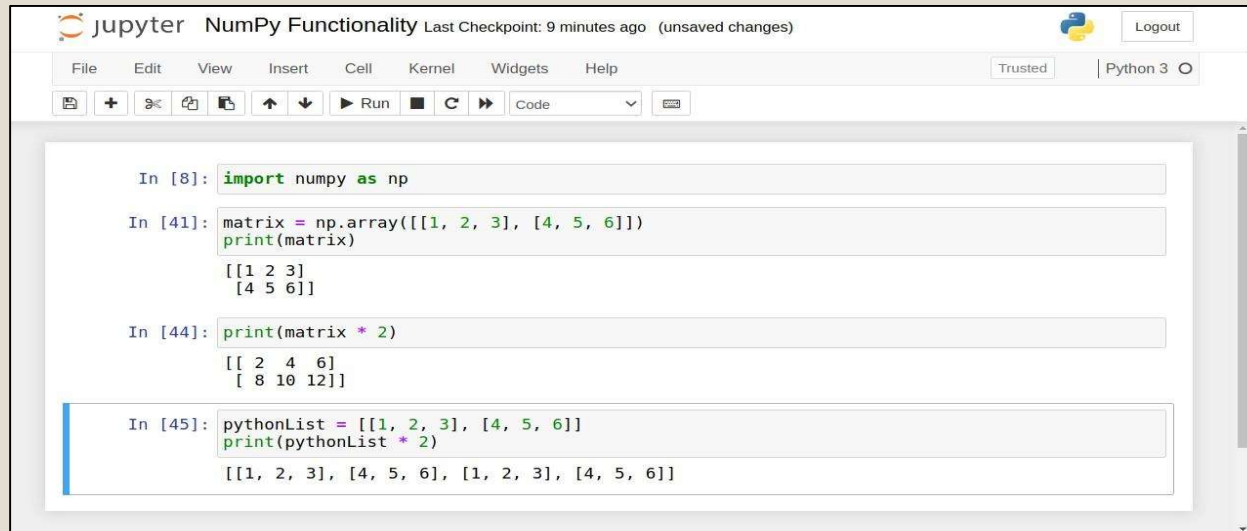
. The third cell, labeled 'In [43]:', defines a Python list 'pythonList' with values [[1, 2, 3], [4, 5, 6]] and attempts to print 'pythonList - 2'. This results in a 'TypeError' with the message: 'TypeError: unsupported operand type(s) for -: 'list' and 'int''. The error message is displayed in red text, and a traceback is shown above it.

```
jupyter NumPy Functionality Last Checkpoint: 8 minutes ago (unsaved changes)
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3
In [41]: matrix = np.array([[1, 2, 3], [4, 5, 6]])
         print(matrix)
         [[1 2 3]
          [4 5 6]]
In [42]: print(matrix - 2)
         [[ -1  0  1]
          [ 2  3  4]]
In [43]: pythonList = [[1, 2, 3], [4, 5, 6]]
         print(pythonList - 2)
         -----
         TypeError                                Traceback (most recent call last)
         <ipython-input-43-0cdc8cdac650> in <module>
             1 pythonList = [[1, 2, 3], [4, 5, 6]]
         ----> 2 print(pythonList - 2)
         TypeError: unsupported operand type(s) for -: 'list' and 'int'
```

Scalar Operations

Multiplication

- We can multiply a scalar with a NumPy array simply by using the (*) operator.
- The scalar quantity is multiplied with each of the elements of the array.
- Note that multiplying a scalar with a Python list will result in list concatenation.

A screenshot of a Jupyter Notebook interface titled "NumPy Functionality". The interface includes a top bar with the Jupyter logo, the title, and a "Logout" button. Below the top bar is a menu bar with options: File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. A toolbar with icons for file operations and execution is located below the menu bar. The main area contains four code cells. The first cell imports NumPy as np. The second cell creates a 2x2 NumPy array named 'matrix' and prints it, showing the output as a 2x2 array. The third cell prints the result of multiplying the 'matrix' by 2, showing the output as a 2x2 array with each element doubled. The fourth cell creates a Python list named 'pythonList' and prints the result of multiplying it by 2, showing the output as the list concatenated with itself.

```
jupyter NumPy Functionality Last Checkpoint: 9 minutes ago (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [8]: import numpy as np

In [41]: matrix = np.array([[1, 2, 3], [4, 5, 6]])
         print(matrix)
         [[1 2 3]
          [4 5 6]]

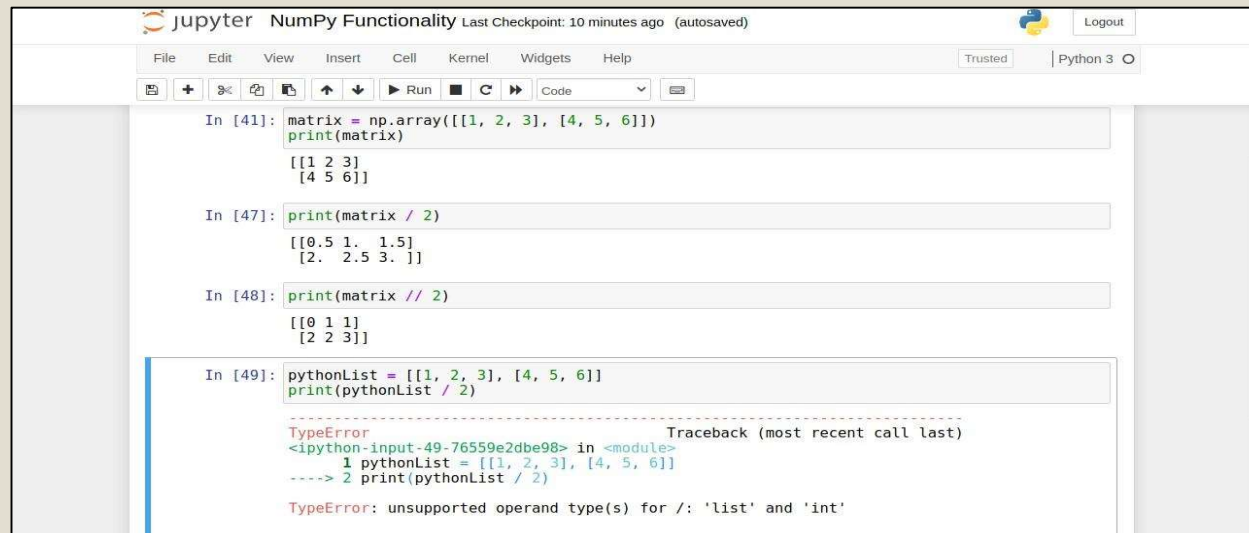
In [44]: print(matrix * 2)
         [[ 2  4  6]
          [ 8 10 12]]

In [45]: pythonList = [[1, 2, 3], [4, 5, 6]]
         print(pythonList * 2)
         [[1, 2, 3], [4, 5, 6], [1, 2, 3], [4, 5, 6]]
```

Scalar Operations

Division

- We can divide a NumPy array by a scalar simply by using the (/) operator for float division or (//) operator for integer division.
- Each of the elements of the array is divided by the scalar.
- Note that dividing a Python list by a scalar will result in an error.



The screenshot shows a Jupyter Notebook interface with the title 'NumPy Functionality'. The notebook contains four code cells. The first three cells demonstrate operations on a NumPy array named 'matrix'. The first cell creates the array and prints it. The second cell divides the array by 2 using the '/' operator, resulting in a float array. The third cell divides the array by 2 using the '//' operator, resulting in an integer array. The fourth cell attempts to divide a Python list named 'pythonList' by 2 using the '/' operator, which results in a 'TypeError: unsupported operand type(s) for /: 'list' and 'int''.

```
jupyter NumPy Functionality Last Checkpoint: 10 minutes ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3
+ - - - - - Run - - - - - Code
In [41]: matrix = np.array([[1, 2, 3], [4, 5, 6]])
         print(matrix)
         [[1 2 3]
          [4 5 6]]

In [47]: print(matrix / 2)
         [[0.5 1.  1.5]
          [2.  2.5 3.  ]]

In [48]: print(matrix // 2)
         [[0 1 1]
          [2 2 3]]

In [49]: pythonList = [[1, 2, 3], [4, 5, 6]]
         print(pythonList / 2)

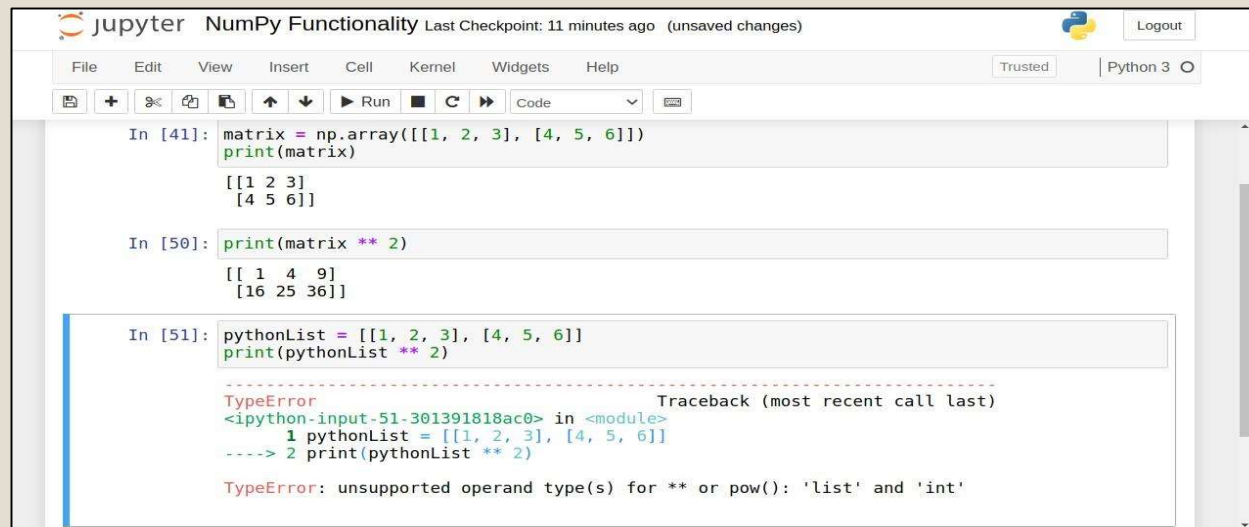
         -----
         TypeError                                Traceback (most recent call last)
         <ipython-input-49-76559e2dbe98> in <module>
         1 pythonList = [[1, 2, 3], [4, 5, 6]]
         ----> 2 print(pythonList / 2)

         TypeError: unsupported operand type(s) for /: 'list' and 'int'
```

Scalar Operations

Power

- We can raise each element of a NumPy array to a power simply by using the (**) operator.
- Note that raising the elements of a Python list using (**) operator will result in an error.



The screenshot shows a Jupyter Notebook interface with the title "NumPy Functionality". The notebook contains three code cells. The first cell defines a NumPy array named 'matrix' and prints it. The second cell prints the square of 'matrix' using the '**' operator. The third cell defines a Python list named 'pythonList' and attempts to print its square using the '**' operator, which results in a 'TypeError: unsupported operand type(s) for ** or pow(): 'list' and 'int''.

```
jupyter NumPy Functionality Last Checkpoint: 11 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [41]: matrix = np.array([[1, 2, 3], [4, 5, 6]])
          print(matrix)
          [[1 2 3]
           [4 5 6]]

In [50]: print(matrix ** 2)
          [[ 1  4  9]
           [16 25 36]]

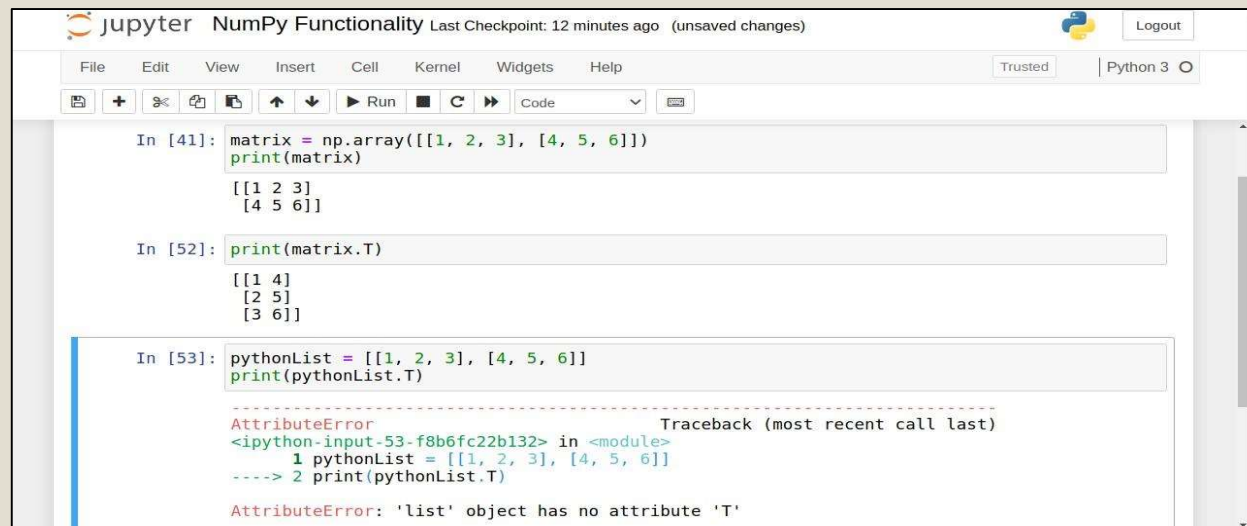
In [51]: pythonList = [[1, 2, 3], [4, 5, 6]]
          print(pythonList ** 2)

          -----
          TypeError                                Traceback (most recent call last)
          <ipython-input-51-301391818ac0> in <module>
              1 pythonList = [[1, 2, 3], [4, 5, 6]]
          ----> 2 print(pythonList ** 2)

          TypeError: unsupported operand type(s) for ** or pow(): 'list' and 'int'
```

Transpose

- We can take the transpose of a NumPy array by putting `.T` at the end of the array.
- Note that taking transpose of a Python list will result in an error.



A screenshot of a Jupyter Notebook titled "NumPy Functionality". The interface includes a top bar with the Jupyter logo, title, and a "Logout" button. Below the top bar is a menu bar with options: File, Edit, View, Insert, Cell, Kernel, Widgets, and Help. A toolbar with various icons for file operations and execution is located below the menu bar. The notebook contains three code cells. The first cell, labeled "In [41]:", defines a NumPy array named 'matrix' with the values [[1, 2, 3], [4, 5, 6]] and prints it, resulting in the output: [[1 2 3], [4 5 6]]. The second cell, labeled "In [52]:", prints 'matrix.T', resulting in the output: [[1 4], [2 5], [3 6]]. The third cell, labeled "In [53]:", defines a Python list named 'pythonList' with the same values and attempts to print 'pythonList.T'. This results in an 'AttributeError' with a traceback showing the error message: 'list' object has no attribute 'T'.

```
jupyter NumPy Functionality Last Checkpoint: 12 minutes ago (unsaved changes) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [41]: matrix = np.array([[1, 2, 3], [4, 5, 6]])
         print(matrix)
         [[1 2 3]
          [4 5 6]]

In [52]: print(matrix.T)
         [[1 4]
          [2 5]
          [3 6]]

In [53]: pythonList = [[1, 2, 3], [4, 5, 6]]
         print(pythonList.T)

AttributeError                                Traceback (most recent call last)
<ipython-input-53-f8b6fc22b132> in <module>
      1 pythonList = [[1, 2, 3], [4, 5, 6]]
----> 2 print(pythonList.T)

AttributeError: 'list' object has no attribute 'T'
```